DEVELOPMENT OF THE NECESSARY SOFTWARE AND COMMUNICATION PATHWAY FOR A COST-EFFECTIVE HIGHWAY TRAFFIC SPEED DETECTION SYSTEM

A Thesis

Presented in Partial Fulfillment of the Requirements for

the Degree Bachelor's of Science in Electrical Computer Engineering

School of The Ohio State University

By: Stephen Thomas Sawyer

*****

The Ohio State University

2006

Thesis Review Committee:

Dr. Benjamin Coifman, adviser

Dr. Keith A. Redmill

Abstract

Gathering data on average highway speed is very important to the field of transportation research. Currently, the most common method of gathering this data is through the use of loop detectors, which essentially consist of metal detectors placed under the roadway. While useful for gathering a wide variety of traffic information, these detectors are expensive and complicated to install.

In this project, an alternative design has been produced, consisting of a bidirectional police radar unit, a communication path, and a computer to store and aggregate the data produced by the radar. The computer uses software that was developed in C for a Linux platform as part of this project. In addition, a webpage was produced on the server that allows visitors to view the data observed by the radar unit in real time, as well as download historical data stored on the server.

Acknowledgments

I would like to thank my advisor Benjamin Coifman for advising this project. His assistance and guidance were instrumental in both creating the opportunity for me to work on this project and keeping me focused on the project.

I thank the Ohio Department of Transportation for providing the equipment, documentation, and communication path necessary for this project, as well as assistance in the unit's deployment.

I thank George Saylor of The Ohio Department of Transportation for his contributions to this project. George's prototype was used as a starting point for the development of the new system, and the documentation and software he provided were instrumental in producing the unit. In addition, he produced the overall concept for the unit.

I thank Matt Graf of The Ohio Department of Transportation for providing a proper communication path, and his assistance in performing a site survey, providing the necessary transportation and assistance in getting the unit deployed. In addition, he was incredibly helpful in setting up the server at the TMC, keeping it going when possible, and letting me come out to the TMC whenever I really messed things up.

I thank Keith Redmill for his highly useful example webpage with graph production, and his assistance in enabling the webpage on the server. The example page he provided is the basis for the current webpage, which saved me a great deal of time.

I would also like to thank Nathan Denning for providing the code to produce the median of a given set of data, his assistance on this project and his commitment to verify the functionality of this unit after my graduation.

I thank Patrick Teets and Bill Triest for their assistance in attempting to get the single board computer to work. While the SBC was not used in the final version of the project, your efforts were still greatly appreciated.

I also thank Alissa Simon for both her assistance in proofreading this thesis, and for her love and support that kept me marginally sane while working on this project.

Last, I thank my parents for teaching me to work hard, supporting me, and enabling me to get this far.

Table of Contents

List of Figures

Introduction

Data on the average speed of highway traffic is useful to detect, locate, and isolate the source of traffic bottlenecks, and can detect potential traffic problems before they become serious. The present method of gathering this data is through the use of loop detector stations. A loop detector is essentially a metal detector that is placed into the roadway. While useful for gathering a wide variety of traffic information [1], loop detectors are expensive and complicated to deploy. The procedure for installing a loop detector in an existing roadway consists of cutting into the pavement, installing the loop itself, then sealing over the loops. As a result of the involved installation process required by loop detectors, each lane must be closed temporarily to allow the installation process to take place. This complex installation process and high cost limit the use of loop detectors to a few critical locations.

To reduce the cost of traffic data gathering, other systems are being developed. The Ohio Department of Transportation (ODOT) developed a prototype of one such system. George Saylor of ODOT designed a system that used a bidirectional police radar gun to calculate average traffic speed in each direction and displayed this average as an overlay on concurrent video footage of the traffic.

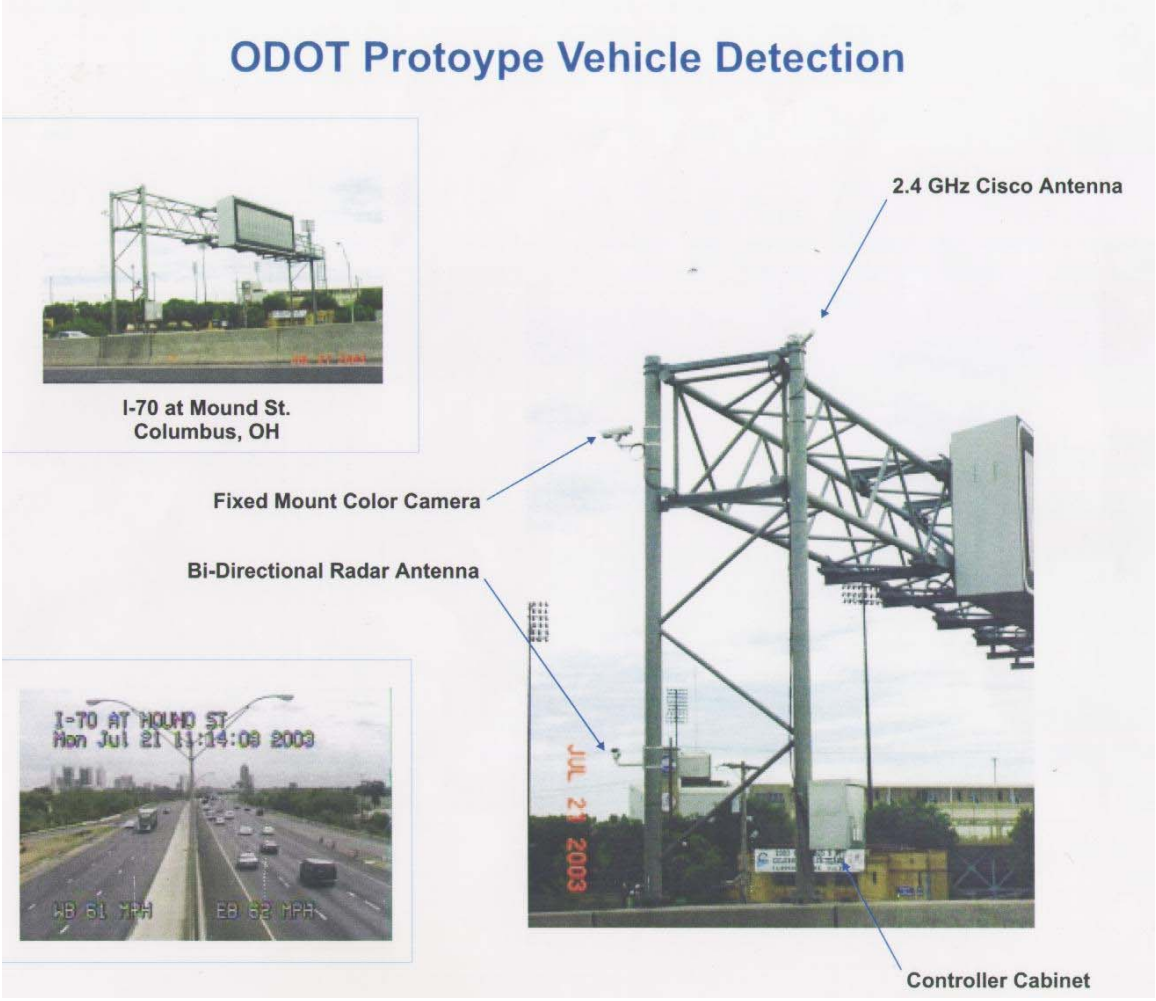


Figure 1: ODOT Prototype Vehicle Detection System (Courtesy of George Saylor)

This prototype system has limitations. First, the unit is located in the median of the road, which can create significant safety issues whenever physical access to the unit is required. Specifically, installing, removing, modifying, and troubleshooting the system become quite complicated, as this requires access to the median of the highway. Second, it does not present the data in a machine friendly format that would facilitate automated control; it only displays the results on the video overlay.

Figure 2: Detailed View of ODOT System Output (Courtesy of George Saylor)

Design Overview

It was decided that a unit would be produced that would be similar to George Saylor's video overlay unit with two major improvements. First, the unit is moved to the side of the road, enabling easier physical access to it. Secondly, instead of sending the data to a video overlay box, the data is sent to a PC that worked as a server. This server aggregates and stores the data from the radar and makes the aggregated data available via a webpage. The system's webpage is visible to anyone on ODOT's intranet, which includes a few computers in OSU's Transportation Research Laboratory in Caldwell Labs. In addition, the unit is designed to be robust enough that it can operate for extended periods of time without requiring human interaction. The resulting radar based design should produces a traffic speed that is accurate enough to enable detection of congested traffic conditions at a drastically lower cost [2].

The MPH Enforcer Radar used in the unit costs $1906, while according to [3] 444 loops were obtained for $166,857. At the site the unit is deployed at, 12 loops are required to cover the highway. Two are needed for each lane, and the road is 3 lanes wide in each direction. Assuming they can be obtained for the same cost as in [3], these 6 loops would cost a total of $4509. The radar detector costs approximately 42.3% of the cost of the loops it replaces.

In addition, the 444 loops in [3] are expected to have an annual cost of $24,601, which would mean 12 loops would expect an annual cost around $388. The annual cost for the radar should require very little work to keep it functioning. However, the estimated costs for the loops here are somewhat generous, as the costs in [3] are 6 years old, and are for a much larger project than our project. In addition, the costs listed only include the loops; the equipment and personnel needed to install the loops were not included, neither was the estimated cost of closing the highway to enable loop installation. It is also worth noting that as the number of lanes increases, the cost of a loop-based system will go up, while the cost of the radar unit will not. Other than the cost of the detectors themselves and the cost of installing the detectors, the systems should have roughly equivalent costs. Both systems have comparable needs with respect to establishing a communication path, supplying power, and housing a server to store the data produced.

Correcting for Error Introduced by the Cosine Effect

In order to make the unit easier to install and safer to maintain, the radar has been moved to the side of the road. Unfortunately, placing the unit on the side of the road introduces a problem with data accuracy. The radar does not measure the true speed of the vehicle; it only reads the projection of vehicle's velocity that is tangential to the radar's line of sight with the vehicle. In other words, the radar reports the absolute speed of the vehicle multiplied by the cosine of the angle between the direction the radar is facing and the direction the vehicle is traveling. If the angle were constant, it would be possible to correct this error by multiplying by a correction factor of $1 / \cos(\theta)$ where θ is the angle between the radar's line of sight and the direction traffic is traveling. Unfortunately, this angle is not constant as vehicles traverse the radar's conical field of view, and correcting for this is nearly impossible, as the radar only reports the speed of the most salient target for both the approaching and receding direction with no indication of where in the cone the target is located. For these reasons, finding the actual angle between the radar's line of sight and the direction of traffic flow is highly difficult. Furthermore should the road where the unit is placed could be curved, this will affect the manner in which the angle to changes as vehicles move down the highway.

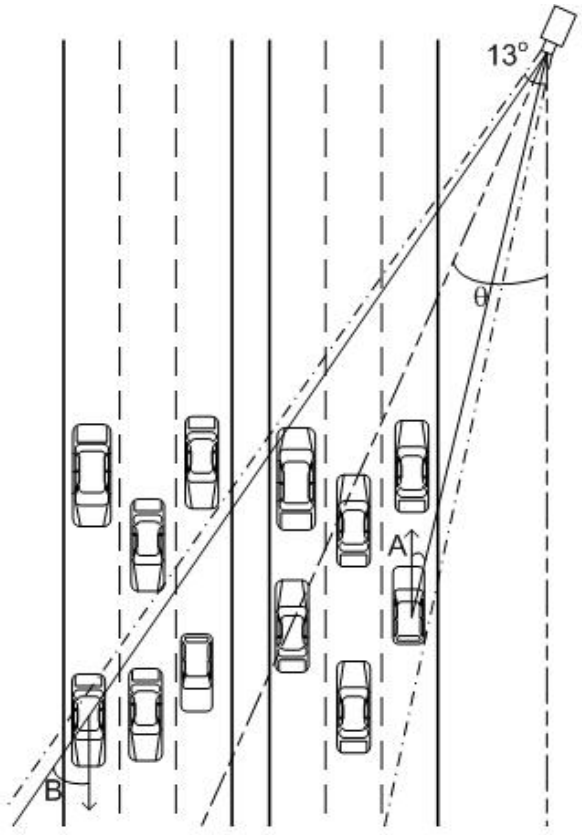


Figure 3: Diagram depicting cosine effect(Produced in Microsoft Visio)

To calibrate the radar to compensate for these unknowns, an automatic calibration algorithm has been developed. The algorithm is based on the fact that traffic will be running at free-flow speeds throughout most of the day. In free-flow traffic most drivers travel at or slightly above the speed limit. The algorithm takes the median speed of all speeds measured over the course of a given day. The median sample should be one for a typical driver, one traveling at or slightly above the speed limit. Next, calculate the correction factor that is needed to set this median speed to the speed limit of 65 miles per hour, then apply this correction factor to every speed in a given set of data. In fact, once an accurate

correction factor is found, that same correction factor can be used across all samples to ensure consistent data correction. However, this algorithm assumes that there is no error introduced by the conical shape of the radar's field of view, as shown in figure 3. The amount of error introduced by the conical field of view depends on the angle θ.

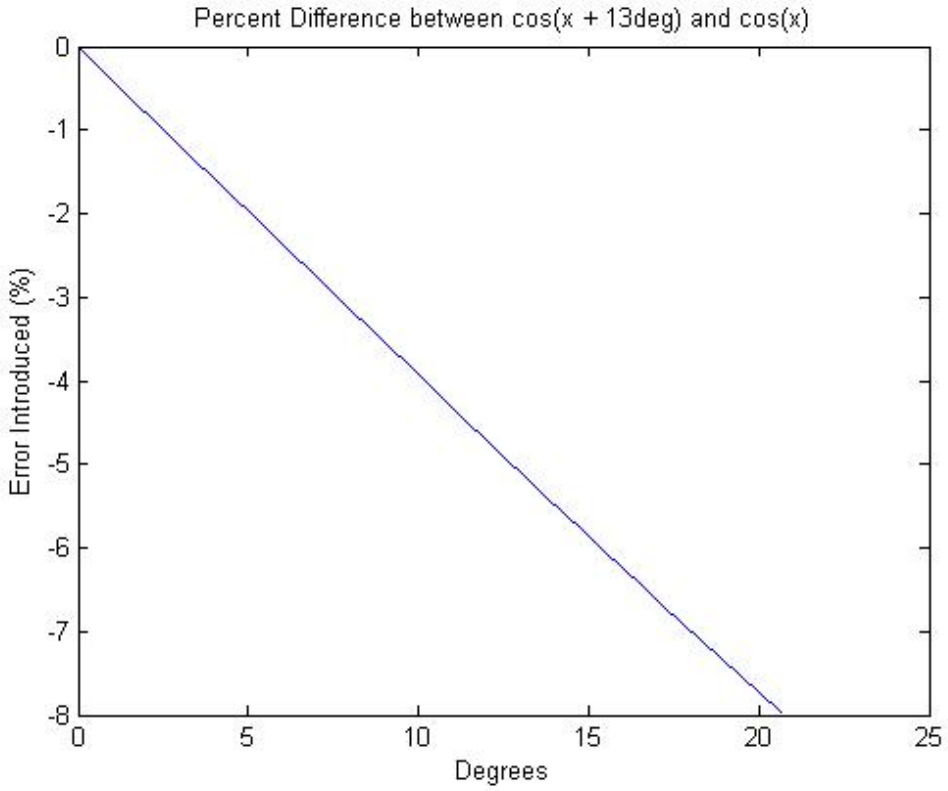


Figure 4: Error introduced over the Cone of Radar (Produced in MATLAB)

As long as the angle between the radar's line of sight and the direction of traffic flow, shown as θ in figure 3, remains less than 20 degrees, the error introduced by the cone will be less than 8% as shown in figure 4. However, this error calculation assumes the same correction factor is applied to both directions. Specifically, if the radar is corrected such that vehicle A in figure 3 will report the

correct speed, vehicle B will have an error such that its corrected value will appear to be approximately 8% smaller than its actual speed. Fortunately, as long as θ remains less than 20 degrees, 8% is the largest error that will be introduced by the conical field of view.

To enable this automatic data correction, software has been developed to extract two separate correction factors from a data file. This software is capable of finding the necessary multiplicative constant needed for the median value for the day to be corrected to an expected value provided by the user. It then reports this correction factor back to the user or, at the user's request; it produces a new data file by multiplying all values in the data by this correction factor. Alternatively, this software can be used to apply a user-specified correction factor to a set of data, enabling the user to find one correction factor and uniformly apply it to all data sets over a long period of time. Once a stable correction factor is found, the program could be made to apply this factor to all data. For the sake of consistency, it is unwise to correct every day to have a median of 65 miles per hour. Inclement weather and other transient events could produce a large shift in the correction factor, and the true values for these speeds would be nearly impossible to recover if a correction factor had already been applied. It is also possible that this correction factor will drift gradually over time for a variety of reasons. For example, it could be caused by gradual shifting of the radar mount. These potential causes for change in the correction factor are the reason the factor is currently applied manually instead of being applied automatically at the time of data collection. Leaving access to the raw data reported by the radar

could be useful in troubleshooting efforts in the future as well as recognizing any drift in the correction factor that could occur over time.

It was decided that θ should be calculated over the course of a day, as a day is long enough to ensure that taking a median will produce the typical vehicle needed for the algorithm, but still short enough that data correction can be accomplished quickly. For greater accuracy in the correction factor, the sample size for the median can be increased, or multiple correction factors can be aggregated.

It is also possible to increase the accuracy on θ by using two separate detectors. For this case, consider two radar detectors A and B looking at the same traffic from different angles. The speeds read by A and B should be equal when they are both multiplied by the proper correction factors, given two or more measurements it is possible to deduce these factors directly. The system deployed as a result of this research only uses one detector. However, it is worth noting that even if the correction factor for $\cos(\theta)$ is incorrect, the speeds will still be directly proportional to the actual speeds, as $\cos(\theta)$ is approximately a constant across all lanes of traffic. As a result it is still possible to detect congestion by noticing a relative drop in the speeds read.

In addition to introducing a cosine effect, moving the radar to one side of the road produces another problem. Specifically, when the radar is on one side of the road, its line of sight to the farther direction of traffic will be blocked from time to time, particularly when a large vehicle passes in front of the detector. As a

result, the data received from that direction will be interrupted more frequently than the data from the nearer traffic. However, even if separate detectors are needed for each direction, the units will still be significantly less expensive than deploying a loop detector station across the freeway.Fortunately, this blocking effect can be drastically reduced by raising the radar unit higher off the ground [4].

Data Gathering and Recording Overview

Recording the data reported by the radar consists of a few separate problems. Speeds must be extracted from the data stream produced by the radar. In addition, these data must be moved to a server where they can be aggregated and stored. The data must then be stored in a manner that is useful for performing additional analysis. Furthermore, while performing these tasks the system must be able to recover from power outages and communication network failures. In order to develop the system software without requiring frequent trips to a field location, software was developed in the Transportation Management Laboratory. The communication link was simulated in the laboratory, and data was generated through use of a tuning fork that causes the radar to report speeds of 50 miles per hour. The software was written in C, and was designed for use in a Linux environment.

Data Acquisition from Radar Unit

The Radar unit reports data over its serial connection in the following format. First, it reports a start transmission byte (STX), which is a byte with the value of “2”, followed by a string of bytes, then an end transmission byte (ETX), which has a value of “3”. In this data stream, the software isolates the approaching and receding speeds, which are the third and fifth bytes of the data stream respectively. These streams are reported at a frequency of approximately 4Hz. It is worth noting that the method used to catch these values is based both on a useful effect of C's read() function and an equally useful effect of the radar. The radar never reports “0” as a value, instead if it was not able to read a speed higher than its minimum threshold of 15 miles per hour it will report “1” as the speed recorded. In addition, the read() function will not return the same byte more than once [5]. The algorithm sets the value of the variable used to read from the radar to 0, then it calls a read() function. If a nonzero value is obtained, this value is recorded then the variable is reset to zero. If the variable remains at a value of 0, then no new data has been obtained, the zero will be ignored and the radar will be read again until a nonzero value is recorded.

The initial development of the algorithm used to collect the data from the radar was derived from the software produced by George Saylor for the video

overlay system. His software was designed to count the number of bytes detected after the start byte (STX) was read. Once this count indicated that it had found the approaching or receding speed, the algorithm immediately performed the all of the processing that could be performed for that particular data point. While this worked fine when only calculating the average speed over 30 seconds and reporting it to a video overlay box, when the algorithm was used alongside the more complex tasks of sending data over the network, storing the data to a local file, and calculating the 30-second median instead of the mean, it failed to consistently catch all of the bytes sent by the radar. Missing even a single byte of data caused the count to be incorrect, so the stream was improperly interpreted. For example, what it thought was the approaching speed might, in fact, be something entirely different. Furthermore, if the start bit or end bytes were missed, the algorithm would have highly unpredictable results.

It was thought that the missed data was caused by the algorithm used to calculate the median. As an attempt to correct this, a separate algorithm was developed that extracted each approaching and receding speed reported by the radar instead of reporting a 30-second the median. However, this algorithm did not behave properly either. Sometimes it would count two approaching speeds before counting a single receding speed, even though the approaching and receding speeds are reported together. Eventually, it was discovered that this was a direct result of the same problem experienced by the median calculating code; intermittently data would be missed, and the count used to identify data cells would become incorrect. As this algorithm was derived from the previous,

not completely working, algorithm, it had inherited the problems, rather than solve them. However, the idea of reporting each of the 4-Hz data points showed promise, so this “live data” version of the algorithm was added into the project.

As another attempt to alleviate these problems, the duties of the program were split between two separate processes that would work in tandem to accomplish the goal of the previous algorithm. The general idea was to reduce the number of samples missed by taking some of the work away from the process that was responsible for reading the data from the radar. Early in its execution, the program executed a fork() system call which created a child process [5]. In an attempt to save functions that had already been written, the child process read data in from the radar, calculated the median for each direction, time stamped the median, and sent the result back to the parent process. The parent process would then take this time stamped median and send it over the network. Again, a similar one-sample “live data” version was produced in parallel. The decision to split the processes in this manner was made because it required little modification to the existing program. As a pipe() system call was used to send the data from one process to the other, the data between these programs was sent as character strings [5]. The previous version of the software produced a character string that it sent using the network code and file storing functions. Instead of calling the file storage and network functions itself, the child process would pass this string into the pipe where it would be read by the parent process. The parent would then handle all the network communication and file storing functions, while the child process would be free to

go back to reading the radar. However, this modification did not address the actual problem within the algorithm. The network transmission code and the file storage code were called shortly after seeing an ETX byte. After the ETX byte is sent, the radar doesn't report anything for roughly a quarter of a second, and it is during this time that the network and file storage routines are run. The problem is the result of the data processing that occurred immediately after a speed was observed. Specifically, it would attempt to update the array used to calculate the approaching median immediately after seeing the approaching speed, instead of waiting for the data stream to end before performing any calculations. As a result, this newer algorithm did not completely alleviate the errors that were occurring. Some samples were still being missed, and as a result, the accuracy of the data was questionable. We did not know if what the program called the receding speed was, in fact, the receding speed. Fortunately, at this point a more in depth analysis of the algorithm was performed, and the true cause of the problem, as previously described, was discovered.

Once it was discovered that the algorithm for reading the data from the radar, one of the larger and more complex sections of code, was erroneous, it was decided that the best solution was to abandon the previous versions and start over. The old algorithms and functions were scrapped, and a new program was written from the ground up. This newer program also consisted of two separate processes. Like the previous version, this program set up a pipe and then forked into a parent and child process. The child process was again responsible for reading data from the radar. However, instead of having the child

process attempt to locate the approaching and receding speeds and immediately perform the necessary calculations before sending them off to the parent process, this newer algorithm has the child read the data from the radar until it detects an end transmission byte. Once it has read the ETX byte, the child process attaches a time stamp to the entire data stream, places the stream and timestamp into a character string, and then sends this data through the pipe to the parent process. The parent process handles extracting the speeds from the data stream reported by the child, and is capable of aggregating these speeds into up to three different types of data records at the option of the user. The program can produce “Median” records, which consist of the median point of data calculated for 30 seconds of data, “live” data records, which consist of every approaching and receding speed reported by the radar, “raw” data records which consist of the time stamped data stream created by the child process, or any combination of the three in parallel. In addition, this software can display the data locally on the screen, store the data in separate local files based on the day, send the data over a computer network to a server program that was developed as part of solving the communication path, or any combination of the above in parallel. Once this algorithm was produced, the variation in the number of samples virtually stopped. Almost all of the 30-second median samples come from sets consisting of 122 or 123 samples of 4-Hz data. However, through inspecting the raw data it became apparent that approximately every 10 minutes, the radar would stop reporting data for about a second and a half, resulting in 117 or 118 samples being reported. Through watching the “raw” data stream in

the lab, this phenomenon was directly observed. The reason the radar stops reporting data in this manner is unknown. However, this loss of data is rather small and should not affect the usefulness of the radar unit.

Communication Path from Radar to the Traffic Monitoring Center

Initially, it was thought the radar would be connected to a small processor that would be located in a cabinet in the field. This small processor would perform the data acquisition duties described in the previous section and then send it over a TCP/IP network connection to a server in ODOT's Traffic Monitoring Center (TMC).

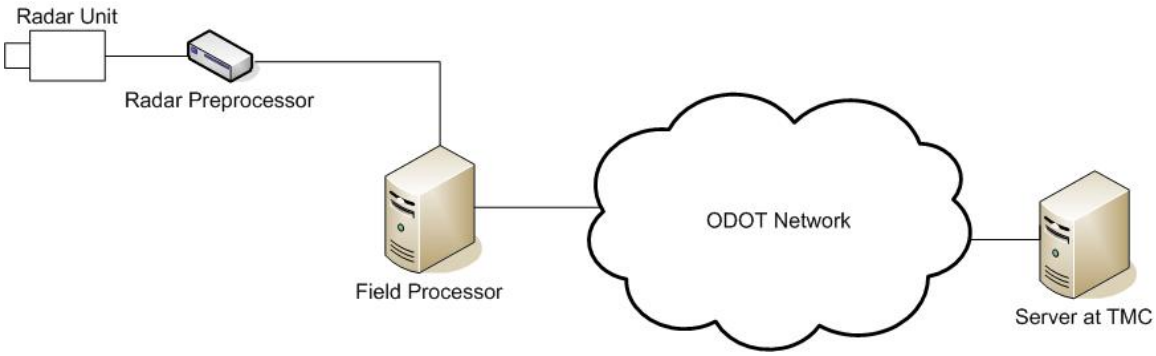


Figure 5: Proposed Communication Path (Produced in Microsoft Visio)

To test the feasibility of this layout, the radar was connected to one computer in a laboratory, which was connected to OSU's ECE computer network; this computer simulated the small processor to be placed in the field with the radar unit, which will now be referred to as the field processor. Another computer on the network simulated the server, which represented the computer located at the TMC. Software was written and tested to perform the task of sending the

radar data from one computer to the other using TCP/IP; the sample code provided at [5] was helpful in programming the communication path.

In simulation, this software showed some potentially serious problems. If the network connection between the field processor and server was disrupted, due to the connection-oriented nature of TCP, the software on both sides locked up. The programs running on the server and field processor both needed to be shutdown in turn, so that they could both be restarted and the connection could be reestablished. The field unit needed to be more robust than this. If someone needed to travel to the field to restart the field processor every time the network failed, the unit would be virtually worthless for long term data aggregation. The strict connection-oriented nature of TCP was undesirable for this system.

In the second attempt at simulating the data transfer, the network portion of the software was changed from using connection-oriented TCP to connectionless UDP, [5] also provides sample code for UDP. There are a few significant trade offs that occurred when the software was changed to use UDP. The desired goal was to make the client and the server software robust enough to operate through network failures. Ideally, once started they should require no further human interaction. Due to its connectionless nature, when the field processor sends data to the server using UDP, it does not need to wait for the server to confirm that it received the data. Instead, the field processor addresses the data to the server, sends it on the network, assumes it will reach its destination and moves on with the rest of its algorithm. Switching to UDP could result in some lost data points, or cause some packets to arrive out of order.

However, this enables the system to operate through network failures without requiring human interaction. In fact, the field processor does not even realize the network is down. This level of robustness in the field processor is highly desirable, and is worth the risk of lost data. Data arriving out of order can be easily corrected by checking the timestamp on each data point.

The UDP system was realized in the lab; however, in a meeting with ODOT engineers, it was decided that the communication link between the field processor and the server would not be a standard computer network. The fiber optic cable that ran out to the sites that were being considered was accessed through fiber modems that had a 25-pin Serial Port that used RS-232 as the communication protocol, and not a true computer network connection that was TCP/IP or UDP/IP compatible. With a functional communication path already present at the sites that were being considered, it was decided that the system should utilize the communication hardware that was already present. An alternate communication pathway, that would utilize this existing fiber network, was selected during this meeting. The interface between the radar and the field processor was a standard 9-pin RS-232 serial port, and the fiber modems used to interface with the fiber network had 25-pin RS-232 ports. Connectors that adapt 9-pin serial to 25-pin serial and vice versa are common, off-the-shelf components. Instead of connecting the radar to a field processor, the 9-pin radar would be connected to a 25-pin serial adapter, which would then connect to a fiber modem, which connects to ODOT's fiber network. This fiber optic connection would be routed to the TMC, where it would first pass through a fiber

modem to convert it back to a 25-pin serial connection. Then it would be converted back to 9-pin serial, where it would connect to the server at the TMC.

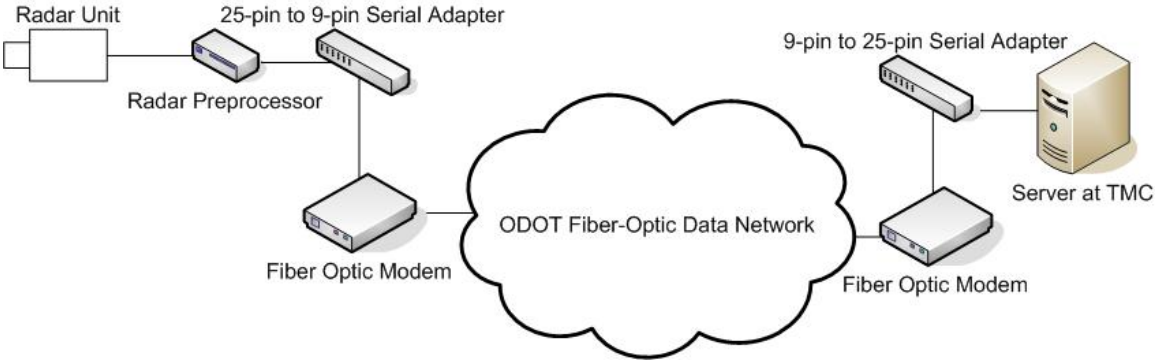


Figure 6: Actual Communication Path (Produced in Microsoft Visio)

Ultimately, this new system consisted of only one computer located at the TMC, with effectively a long, mostly fiber optic, serial cable as shown in figure 6. This new system afforded some significant advantages. First off, the field processor was eliminated altogether, and ODOT's fiber network was ultimately being used as an incredibly long serial cable. This means the old field processor software could be used as the basis for the server software in the new system, as the server would be reading the raw radar data in from its serial port, just as the field unit would do under the older system. In this communication path, the server software developed for the proposed communication path is unused.

At this point, the algorithms for sending the data over a UDP/IP network were complete and ran properly in the lab. In the event that this system is desired, in a situation where the radar cannot be connected straight to a fiber network, the UDP/IP connection could provide a method for getting the data from a field processor in real time, possibly through using wireless network hardware.

Data Record Formatting

The “raw” data has the simplest format. The data records for the raw data have the following format:

R<DAY,MM/DD/YY,hh:mm:ss> A B C D E F

In this format, R is a placeholder that indicates that the field is raw data. DAY is the three-letter abbreviation for the day of the week. The day of the week was included in the timestamp as traffic patterns should be different on weekends than they are on weekdays. As a result, the day of the week could be potentially useful information. Next MM/DD/YY, contains the date the data were recorded, in the order of Month/Day/Year, with two numbers for each field. After the date, the time is recorded in the format hh:mm:ss, for hours, minutes, and seconds respectively, again two digits for each field. Last, the raw data record lists the six consecutive values reported by the radar at this time. Note that if everything is being read properly, A should have a value of 2 and F should have a value of 3, as these are the STX and ETX bytes respectively. Also, note that C should be the approaching speed and E should be the receding speed.

The program can also store “live” data records. These records have the following format:

L<DAY,MM/DD/YY,hh:mm:ss> A_val: AAA R_val: RRR

The L at the beginning of the line indicates that this record is a “live” data record. Specifically, there is one live data record made for each sequence received by the server. The date and time stamp for this file are recorded in the same manner as they are in the “raw” data file. The values AAA and RRR are the approaching and receding speeds measured by the radar, respectively. Note that these values always take up three characters. In the event of an error in the data stream, the values for AAA and RRR will be recorded as LOST, with the L placed immediately after the colon after A_val and R_val, meaning the record will have the form:

L<DAY,MM/DD/YY,hh:mm:ss> A_val:LOST R_val:LOST

If this occurs, it means the data stream does not have the expected number of bytes. In the laboratory, this occurs when the radar is connected or disconnected from the computer while the process is running. In the field, this could occur should part of a data stream be lost in transit on the fiber network.

The most complex data record is the “Median” data record. These records report the median speed over periods of, typically, 30 seconds of data:

M<DAY,MM/DD/YY,hh:mm:ss> A_med: AAA (ANOZ/ATOT) R_med: RRR (RNOZ/RTOT)

Like the other forms, this one begins with the character M to designate the record as a median record. Next are the date and time stamp, which is the same as it is in the “raw” and “live” data records. After this, the value AAA will be

replaced by the median speed recorded by the radar over the sample period. After the median, the record shows ANOZ and ATOT, which report the number of nonzero samples recorded by the radar, and the total number of samples reported by the radar for the approaching direction. Next, this pattern is repeated for the receding traffic with the values RRR, RNOZ, and RTOT. These are the median speed, number of nonzero samples, and the total number of samples read respectively. While these records are calibrated to typically contain 30 seconds of data, they actually contain the data reported by the radar until the timestamp shows 00 or 30 in the seconds field of the timestamp. This distinction is most noticeable when the program is started; the first record will almost always report fewer samples.

Recovery from Power Failures

Enabling the system to recover from power failures was simple. Specifically, once the power is restored to the computer, it will reboot. The program needs to be restarted once the computer is booted. To cause the software to be automatically run as part of the boot procedure the command for executing the program must be added in the /etc/init.d/rc.local file. This script is the last script executed by the computer on boot up, ensuring that any systems, such as network functionality, that the system might need will already be running. However, adding only the command to start the program will cause the files to be stored, by default, in the root directory of the system. To store the data elsewhere on the system, enter a cd <directory> command into the rc.local script before executing the software. The current implementation places the data files in the directory /var/www/html/data, where they can be accessed by users through the webpage.

However, it is worth noting that if the software is using the UDP/IP communication software and loses power and network capability, and power is restored before the network connectivity, an odd situation can occur. If the computer is using DHCP it will not be able to successfully connect to the remote server without properly setting up its own network connection first. If the

software is unable to locate the server, it will make a system call to the network startup script periodically, to attempt to reinitialize the computer’s network connection. Once the network returns, this enables the system to reinitialize its network connection automatically, which enables the field unit to send data to the server. It is worth noting that the exact name and location of the script to restart the network can vary from one distribution of Linux to another. The line of code that makes the system call to this script may need to be changed if the software is run on another system. Fortunately, while this particular complication is handled by the software, it is not needed in the present implementation, as the radar program's network code is unused for this particular unit.

Robustness to Operate Through Network Failures

Prior to switching the design to using the serial fiber modems, Network Failures would be dealt with by using UDP. As explained in the section on the communication path, UDP enables the system to run on a network without a network failure causing the program to lock up.

While the fiber-optic ODOT network used by the unit should easily meet the needs of the 4Hz radar data, it is possible that some portion of the data sent over the fiber will be lost. If a given sample does not report the proper number of bytes, a speed will not be recorded, instead the record for that particular cell of data will be given a value of LOST, as described in the Data Record Formatting section. If the radar does not report a speed for 10 seconds, the program assumes there is a problem with its serial connection to the radar. It automatically closes and reopens the serial port, which effectively restarts the program's use of the serial port. When randomly unplugging the serial port occasionally caused the program to lock up in the laboratory, this solution enabled the system to recover data flow shortly after the radar resumed communication, without restarting the program.

Viewing the data recorded by the Radar unit

A webpage has been set up on the server that is storing the radar data. This page can be viewed by pointing a web browser to http://66.145.155.87/ from any computer on the ODOT intranet. At this page, users can view a graph of raw traffic data stream, download traffic data files, and request to view a graph for historical data that is held on the server. At present, the OSU Transportation Systems Laboratory has the access necessary to view the page.

The website itself was made by writing the HTML from scratch. The information in [6] was helpful in getting the dialog box to at the bottom to operate properly. However, creating the graphs was notably more complicated. The graphs were produced using PLplot [7], a low-level graphing program. Effectively, a C program was written which calls PLplot functions to draw the graph. These functions are linked to PLplot at compile time. This program is then stored in the cgi-bin directory of the webpage. Next, the page calls the program as if it were an image. This call starts the program, which produces a graph in the form of a PNG image. This image is then displayed on the webpage. There are a total of four different versions of the graphing program running on the webpage, all of which use the 30-second median file.

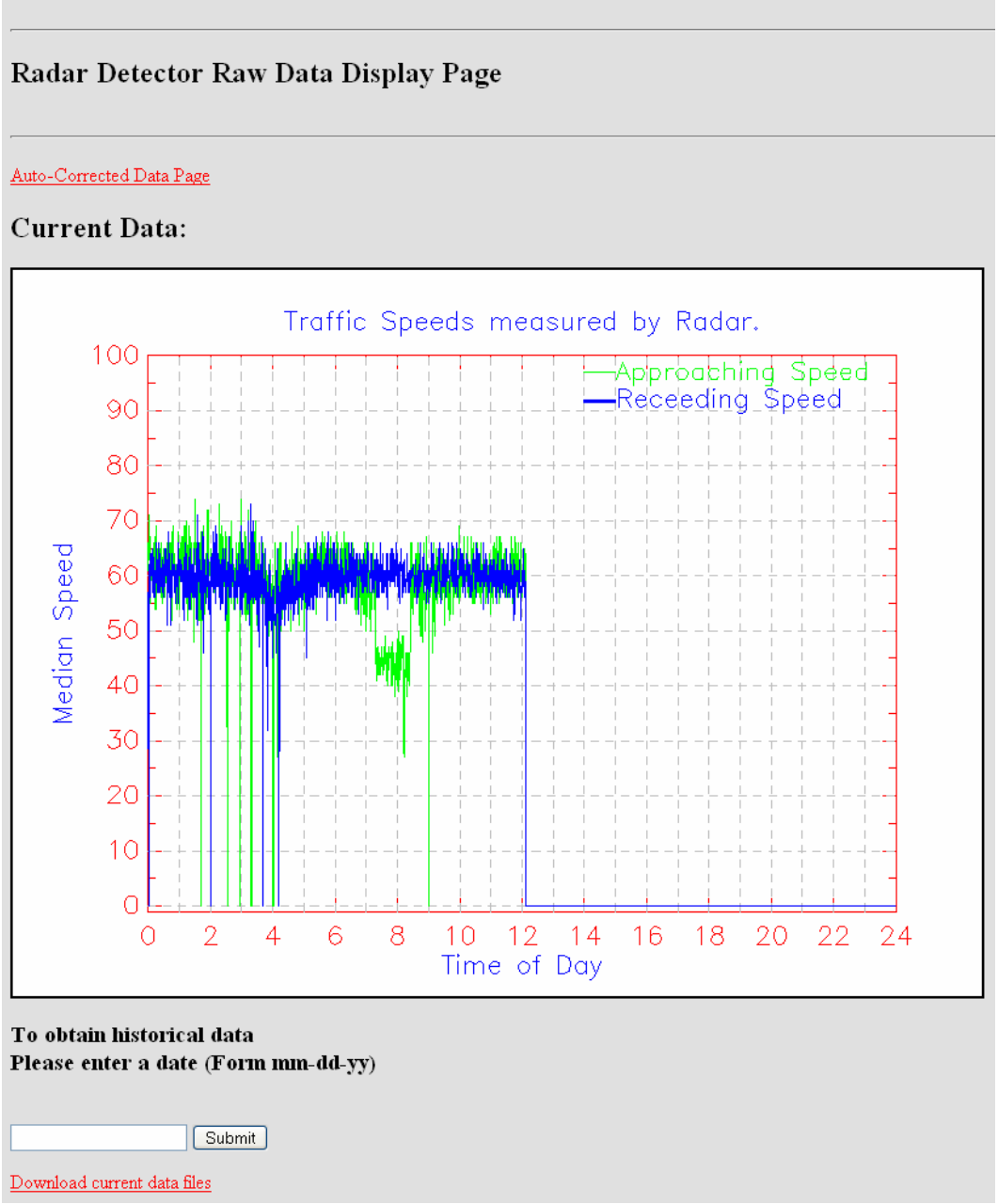


Figure 7: Screen Capture from Website on TMC Server

One version of the graphing code is shown directly on the raw data website. This program, ShowCurrent, produces the graph for the current date, based on the 30-second median data file for the current day. The second program, ShowGraph, is called whenever the visitors to the site request a

historical graph from the dialog box at the bottom of the site, this version uses the 30-second median data file as well. This different version was needed to handle the arguments the user types in the dialog box properly. The third and fourth versions are ShowCurrent-corrected and ShowGraph-corrected, these are analogous to ShowCurrent and ShowGraph, except they are run by a separate page called “corrected.html”. These versions apply a multiplicative correction factor to force the median for the whole day to be 65 miles per hour to cause them to more accurately reflect the actual traffic speeds at the site.

The website also contains a direct link to the directory where the data is stored. This enables visitors to the webpage to download the data files produced by the radar software for analysis.

Considerations for Site Selection

Even if the radar system can gather the traffic data and store it in a manner where it is useful for data analysis, the data is meaningless if it cannot be shown to be accurate. In order to enable the system to be tested for data accuracy, the radar should be deployed in a location where the speeds the radar reports can be compared to speeds that are known. As a result, it was decided that the radar would be deployed near a dual loop detector station. Deploying near a dual loop detector station allows the radar's speeds to be compared to a known speed measurement device. If the radar and the dual loop detector station show comparable speeds for traffic, and can be demonstrated to show the same trends in traffic speed, that should be sufficient to prove that the system is functioning properly.

In addition, the radar needs sufficient space between the actual unit and the traffic it is measuring. There needs to be enough room for the 13-degree radar cone to be able to cover a significant amount of the traffic moving in each direction, preferably all of it. The wider the road is, the more space needed between the radar and the section of highway it is detecting.

Last, the radar needs to be deployed at a location where it can be tested for a wide variety of speed ranges. Ultimately, the speed detection is desired as a method of detecting traffic congestion. To test the system's ability to detect congestion, the system will be deployed in an area where traffic is congested on a regular basis. This requirement enables us to verify that the system is doing the job that it is intended to do.

Site Selection

In light of all these considerations, the unit was deployed on I-71, between 17th Ave and Hudson St. This location contains a dual loop detector station, easy access to communication and power, and sees congestion in both directions. It is also a location where the radar could be installed without a great deal of difficulty. Appendix A contains an as-built schematic of the site selected from Transdyn Controls. The radar was installed next to station V0007 on the northbound side of the freeway, as shown in Figure 8.

Figure 8: Radar Unit Field Deployment Photograph

The radar unit is located at the top of the pole in the middle of the picture, and is pointed at the barrier wall between the northbound and southbound traffic. The radar is facing north; as a result the northbound traffic is read as the receding traffic, while the southbound traffic is read as the approaching traffic. Over the course of verifying the accuracy of the unit, the radar will be turned to face the southbound direction. This will enable the system to be verified under different visibility conditions, as well as with a different angle between the flow of traffic and the radar's line of sight.

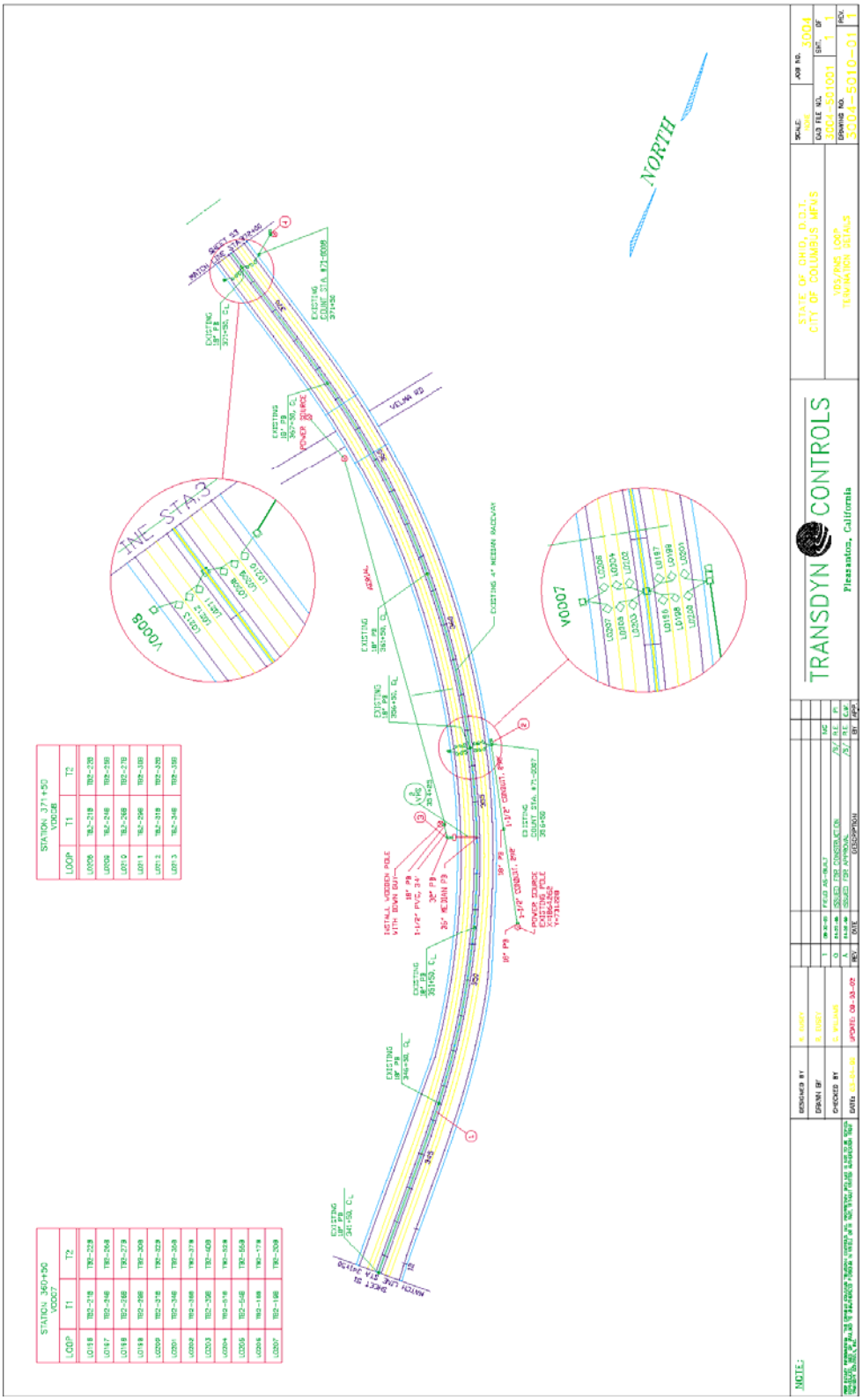
To cover all three lanes of traffic in each direction with the radar's 13-degree cone, the radar needs roughly a minimum of 400 feet of distance between where its location and the traffic it is measuring. This site has more than enough visibility in both the northbound and southbound directions, even though it contains a curve. The curve of the road at the site could produce some data inaccuracy but it is important to demonstrate that the detector can detect congestion under a less than ideal situation

Conclusion

This project has progressed much slower than expected, due to problems with prospective field platforms, as well as the time consuming process of software and system development. One of our early choices for a field platform failed, and there were significant problems with many of our secondary choices for a field unit. In the end, it turned out that the redesign of the communication path removed the need for a field processor. Had this communication path been selected before work on the software began, the process of developing the software for the UDP/IP network could have been bypassed, saving a great deal of development time. In spite of these changing plans, the project is on a path toward success. At the time of this writing, the radar system has been deployed and has gathered about a week of data. The data appears to be accurate at a first glance, but has not been examined in detail. The screenshot shown in Figure 7 was taken on Wednesday, March 15th, 2006, shortly after noon. The data shown on the graph appears to be reasonable and seems to indicate traffic congestion in the approaching, or southbound, direction from around 7:30am until 9am, which is reasonable given its location.

Unfortunately, due to time constraints I will not be able to personally verify the functionality of the radar system. That task will mostly be handled by Nathan Denning, a fellow OSU student and colleague on this project. Hopefully, this new tool will enable ODOT engineers to gather traffic speed data in a quick and cost-effective manner that will allow them to monitor traffic conditions cheaply, and help improve to freeway conditions across the state.

Appendix A: As-Built Schematic Showing Site (from Transdyn Controls)



References

[1] Chen,C; Petty, K; Skabardonis, A; Varaiya, P; Jia Z (2001) “Freeway Performance Measurement System: Mining Loop Detector Data” *Transportation Research Record. Issue 1748.* Transportation Research Board, Washington DC. 96-102.

[2] Gates, T; Schrock S; Bonneson, J (2004) “Comparison of Portable Speed Measurement Devices” *Transportation Research Record. Issue 1870.* Transportation Research Board, Washington DC. 139-146.

[3] Carter, M.; et. al. (200) “Metropolitan Model Deployment Initiative: San Antonio Evaluation Report (Final Draft), Section 4: Improved Traffic Management, Subsection 4.3.1 Freeway Management Expansion Costs”. San Antonio, TX. <http://www.itscosts.its.dot.gov/its/benecost.nsf/0/C49E7F617F4D605C85256DDE005ED847?OpenDocument&Query=CApp>

[4] Franklin, C (2005) “Practical Issues when considering non-embedded traffic detection technologies.” *IMSA Journal Vol 43, Issue 3.* International Municipal Signal Association, Newark NY., 60-63.

[5] Galvin, Silberschatz (2002) *Operating System Concepts*, 7th Edition.

[6] “Sockets Tutorial” <http://www.cs.rpi.edu/courses/sysprog/sockets/sock.html>

[7] “HTML Form Tutorial.” Simfatic Solutions. <http://www.javascript-coder.com/html-form/html-form-tutorial-p1.phtml>

[8] LeBrun, Maurice; Furnish, Geoff. (2005) “The PLplot Plotting Library Programmer’s Reference Manual”. <http://plplot.sourceforge.net/docbook-manual/plplot-html-5.5.3/>